Confiabilidade de Sistemas Distribuídos Dependable Distributed Systems

DI-FCT-UNL, Henrique Domingos, Nuno Preguiça

Lect. 1b
Introduction

2015/2016, 2nd SEM

MIEI
Mestrado Integrado em Engenharia Informática

Outline

- Concepts, Terminology / Dependable Systems and Dependability Criteria
- Fault-Tolerance vs. Intrusion Tolerance
- Failures/Attacks Masking and Techniques
- Failure Detection vs. Intrusion Detection

Dependable Systems

- Concepts, Terminology
- Dependability Criteria

What is "Dependability" ?

- Context:
  - A component provides services to clients.
  - To provide services, the component may require the services from other components

    → a component may depend on some other component.

    We say that a **component C depends on C*** if the **correctness of C's behavior depends on the correctness of C*'s behavior**.

  What are "these" components about ?

Dependable Distributed Systems

What are components about ?

In Dependable Distributed Systems components are (generally):

- Processes (Computations + Data-Processing)
- Channels

Dependability Properties

Base dependability properties

- **Availability**
  - Readiness for usage
- **Reliability**
  - Continuity of service delivery
- **Safety**
  - Very low probability of catastrophes
- **Maintainability**
  - How easily can a failed system be repaired

Dependability Properties

Base dependability properties

- **Availability**
  - Readiness for usage
- **Reliability**
  - Continuity of service delivery
- **Safety**
  - Very low probability of catastrophes
- **Maintainability**
  - How easily can a failed system be repaired

Availability and Fault-Tolerance and Conditions

Reliability vs. Availability (1)

- Reliability *R*(*t*):
  - **probability** that a component has been **up and running** (correctly and continuously) in the time interval ***[0, t]***

Conventional Metrics:

- **MTTF:** Mean Time To Failure:
  - Average time until a component fails
- **MTTR:** Average time it takes to repair (recover) a failed component.
- **MTBF:** Mean Time Between Failures
  - MTTF + MTTR

Reliability vs. Availability (2)

- Availability: A(t):
  - **Average fraction of time** that a component has been up and running in the interval ***[0 , t]***
- Long-Term Avaiability (or Always Available):
  - ($A(\infty)$

Relating:

- **A = MTTF/MTBF**

  => A = MTTF / (MTTF + MTTR)

Reliability vs. Availability (3)

- Important Observation:
  - Reliability and availability make sense:
  - **If we have an accurate notion of what a failure actually is**
  - **Requires a "very well-defined" Failure Model, related to the System Model and Design**

  - => Reliability vs. Availability Tradeoffs BY DESIGN !

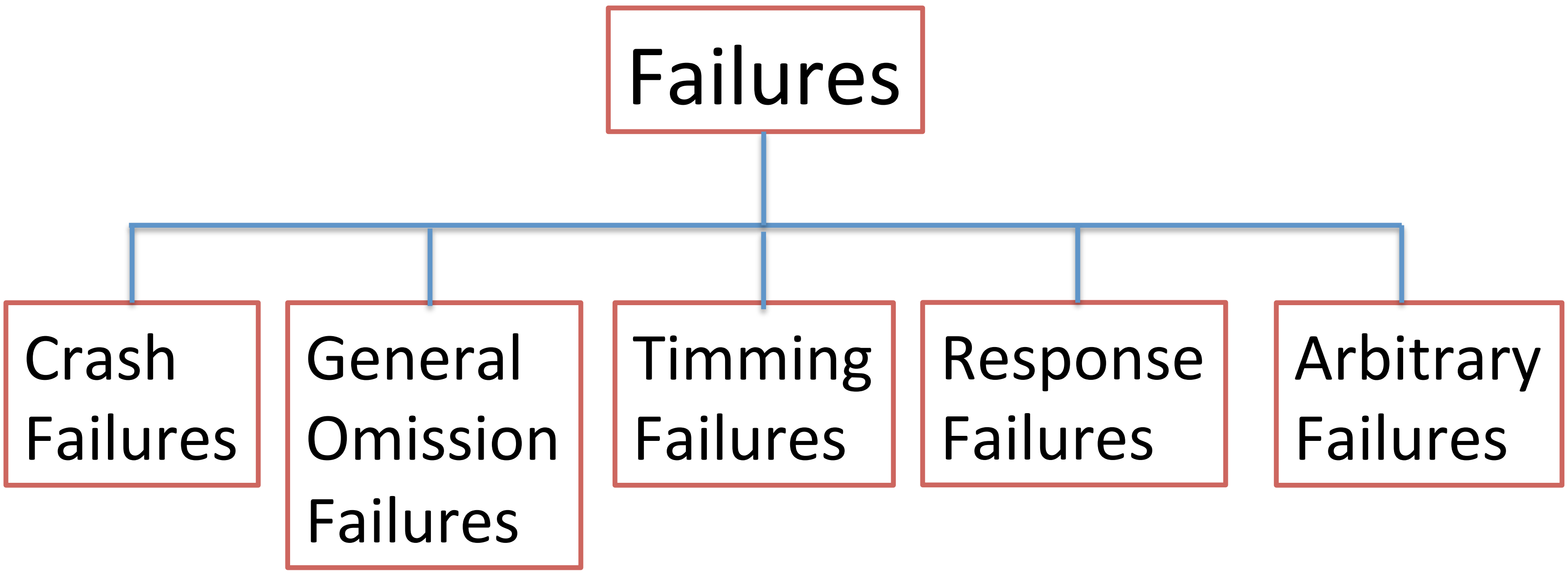
Terminology: Let’s start by Failures

Term	Description	Example
Failure	May occur when a component is not living up to its specifications	A crashed program
Error	Part of a component that may lead to a failure	A programming bug
Fault	The cause of an error	A sloppy programmer

Terminology: Let’s start by Failures

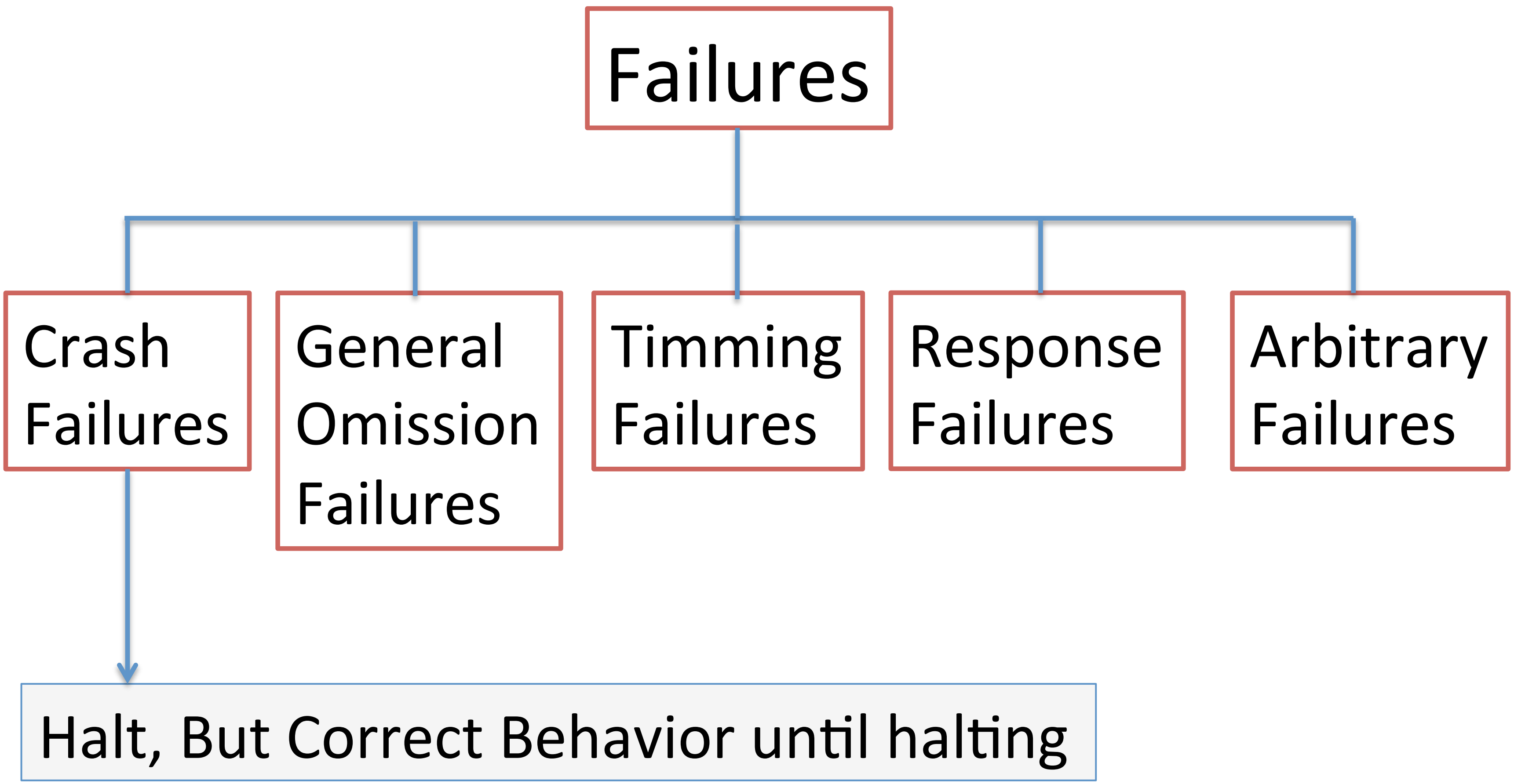
Term	Description	Example
Fault prevention	Prevent the occurrence of a fault	Don't hire sloppy programmers
Fault tolerance	Build a component such that it can mask the occurrence of a fault	Build each component by two independent programmers
Fault removal	Reduce the presence, number, or seriousness of a fault	Get rid of sloppy programmers
Fault forecasting	Estimate current presence, future incidence, and consequences of faults	Estimate how a recruiter is doing when it comes to hiring sloppy programmers

Failure Models
Generic Characterization



Typology (as ref. in Andrew Tanenbaum, Maarten Van Steen,
Distributed Systems - Principles and Paradigms, Chap. 7 – Fault Tolerance (2nd Edition,

Failure Models
> Crash Failures



Failure Models
> Omission Failures

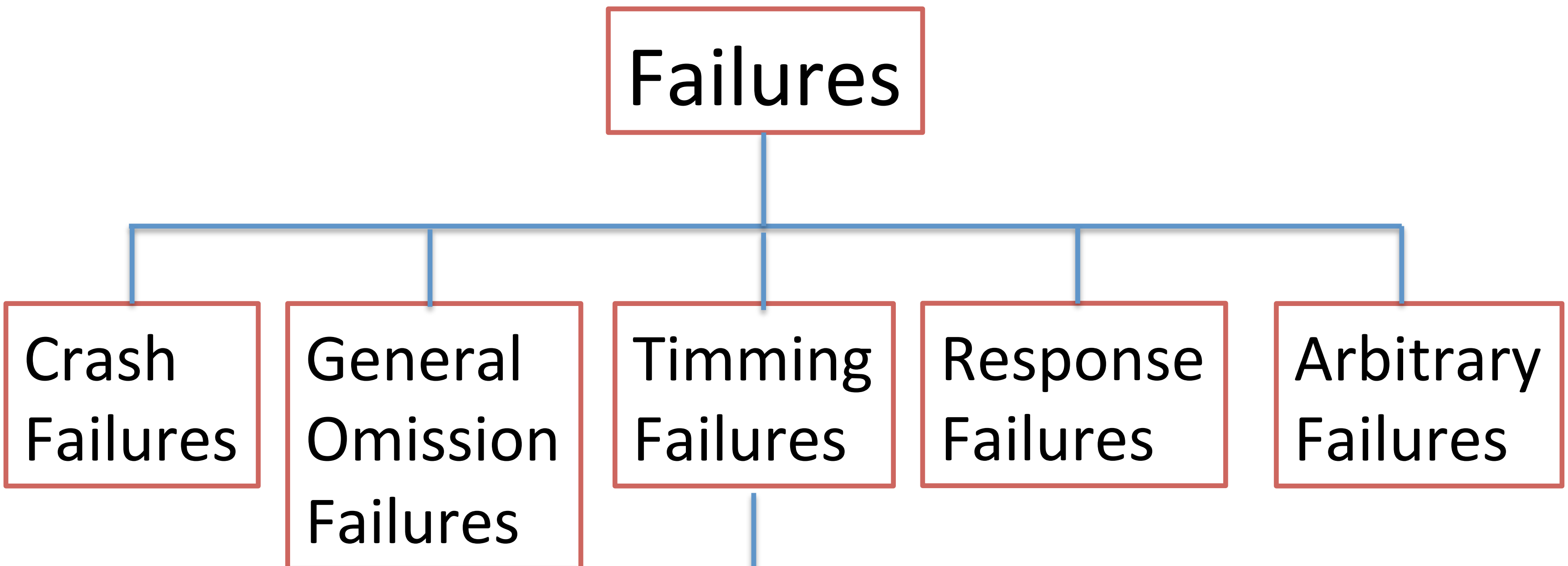
- Failures
  - Crash Failures
  - General Omission Failures
  - Timming Failures
  - Response Failures
  - Arbitrary Failures

Failure in Sending or Receiving Messages

Recv Omissions: Correctly Sent Messages are not Received

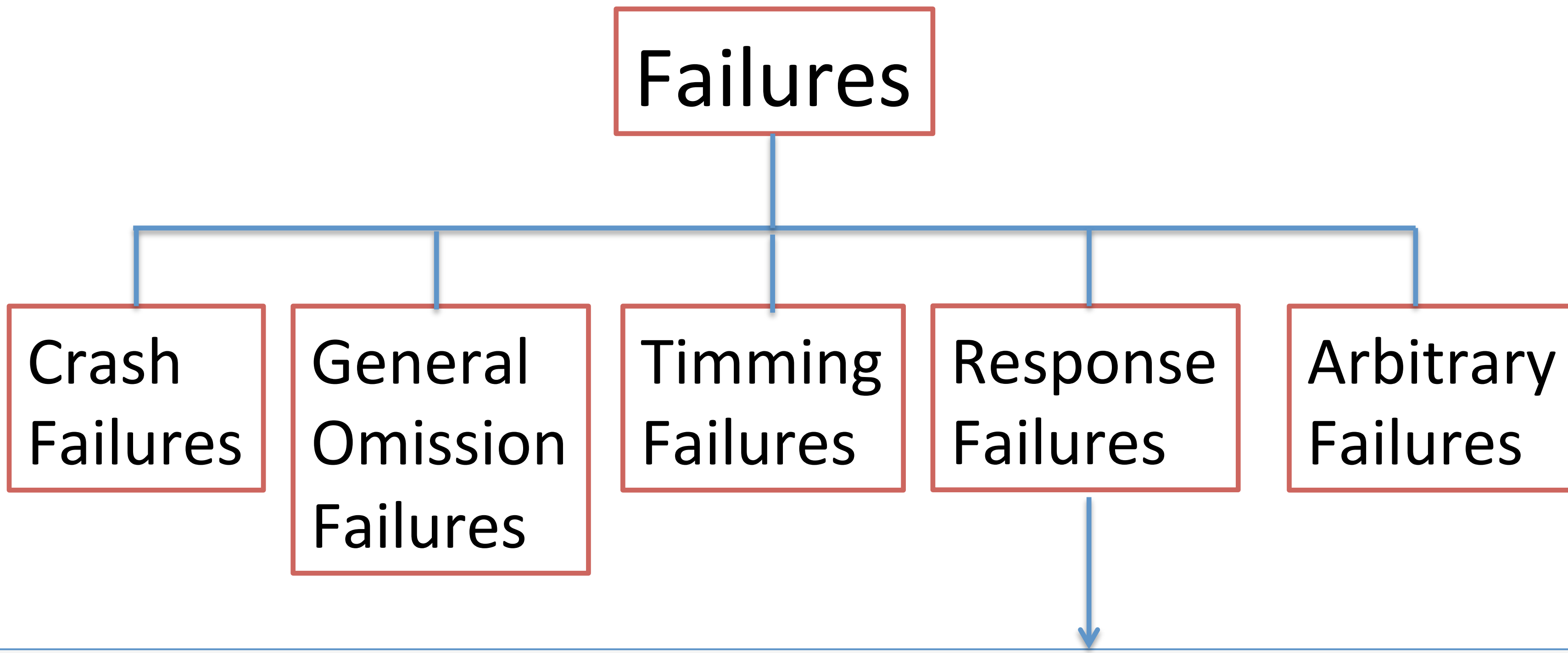
Send Omissions: Messages not sent correctly (that should have)

Failure Models
> Timing Failures



Correct Output, but provided by outside a specific time interval
Performance Perceived Failures: Component Answer too Slow

Failure Models
> Response Failures



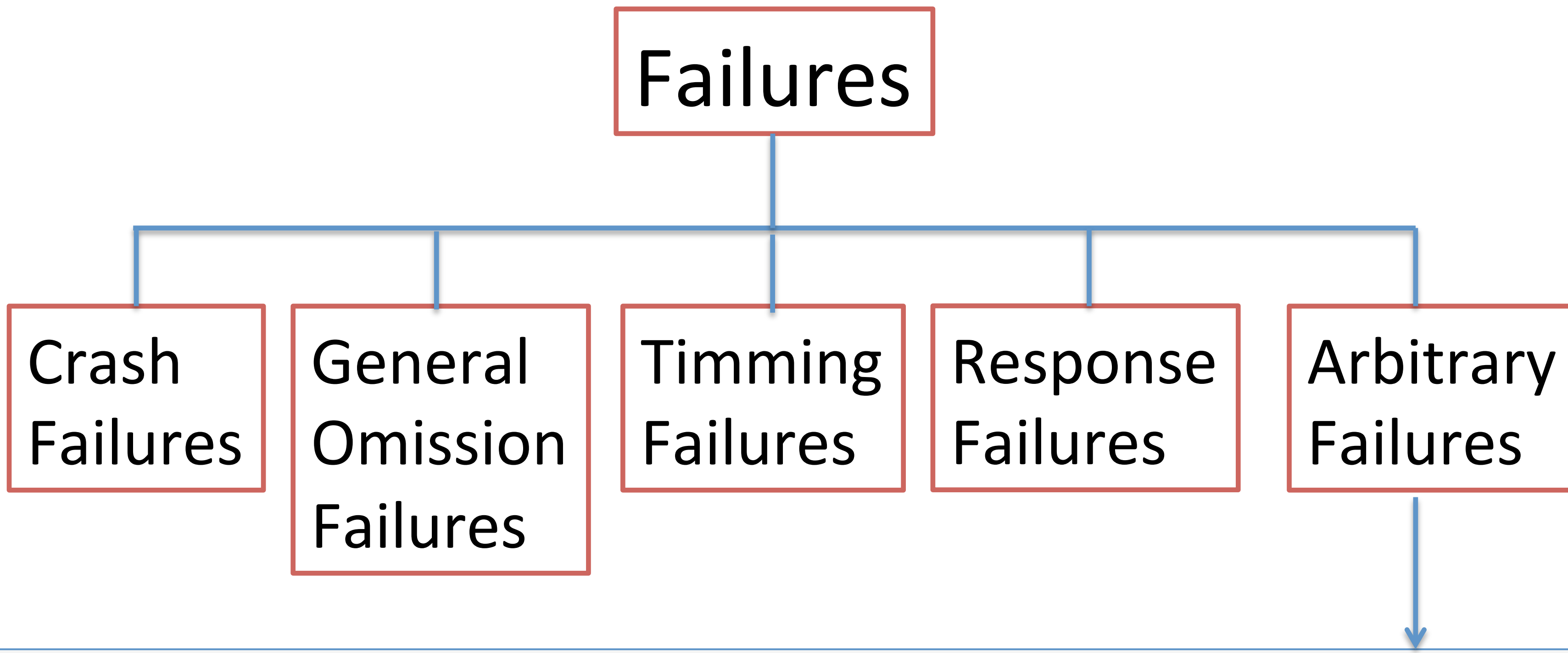
Incorrect output, but cannot be accounted to another component

> **Value Failures**: wrong output values

> State-Transition Failures: deviation from correct flow of control

(Note: this failure may initially not even be observable)

Failure Models
> Arbitrary Failures

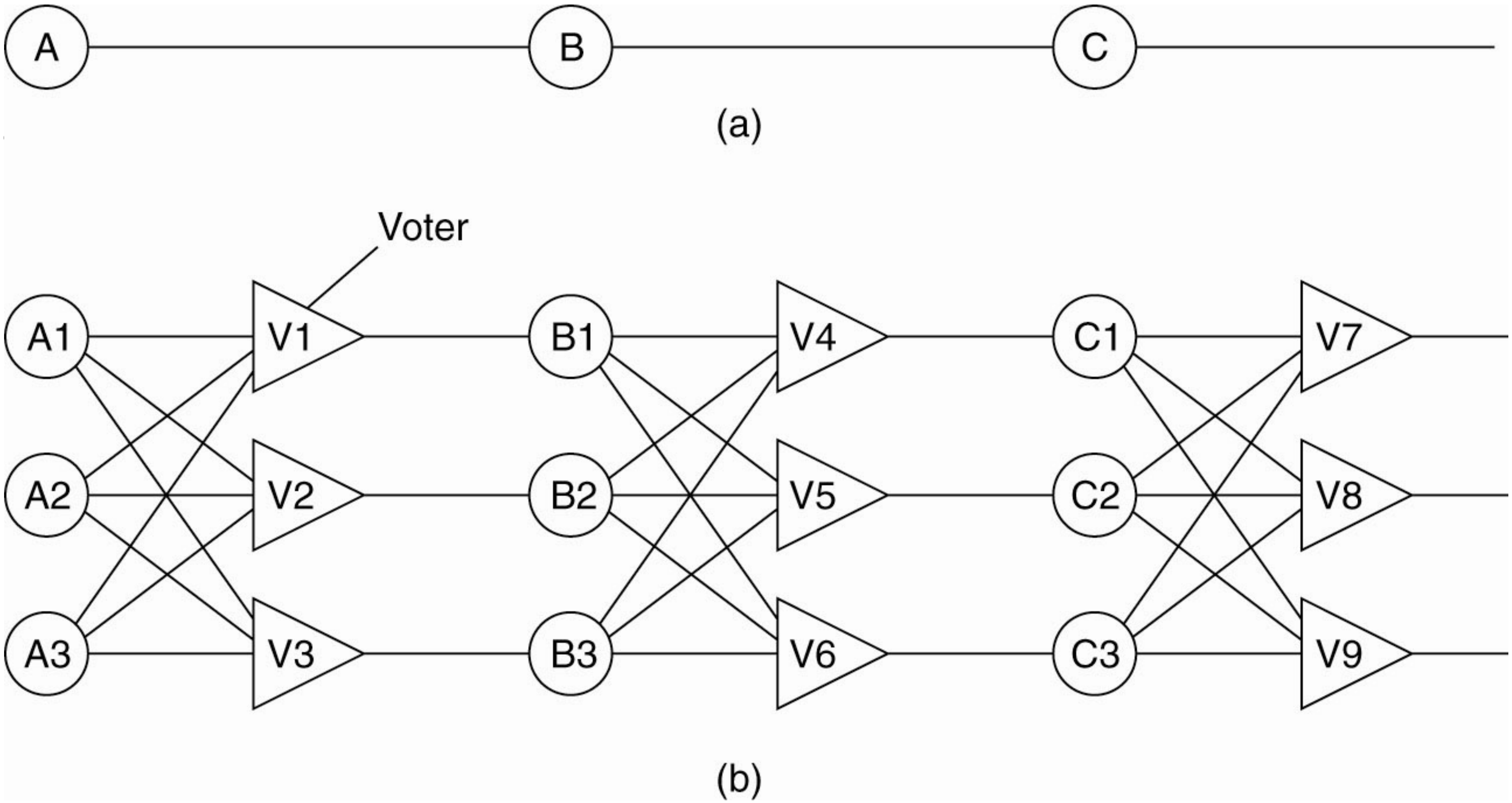


Any (or any combination of) failure may occur, perhaps even unnoticed (silent failures) or not (noticed or detectable failures)

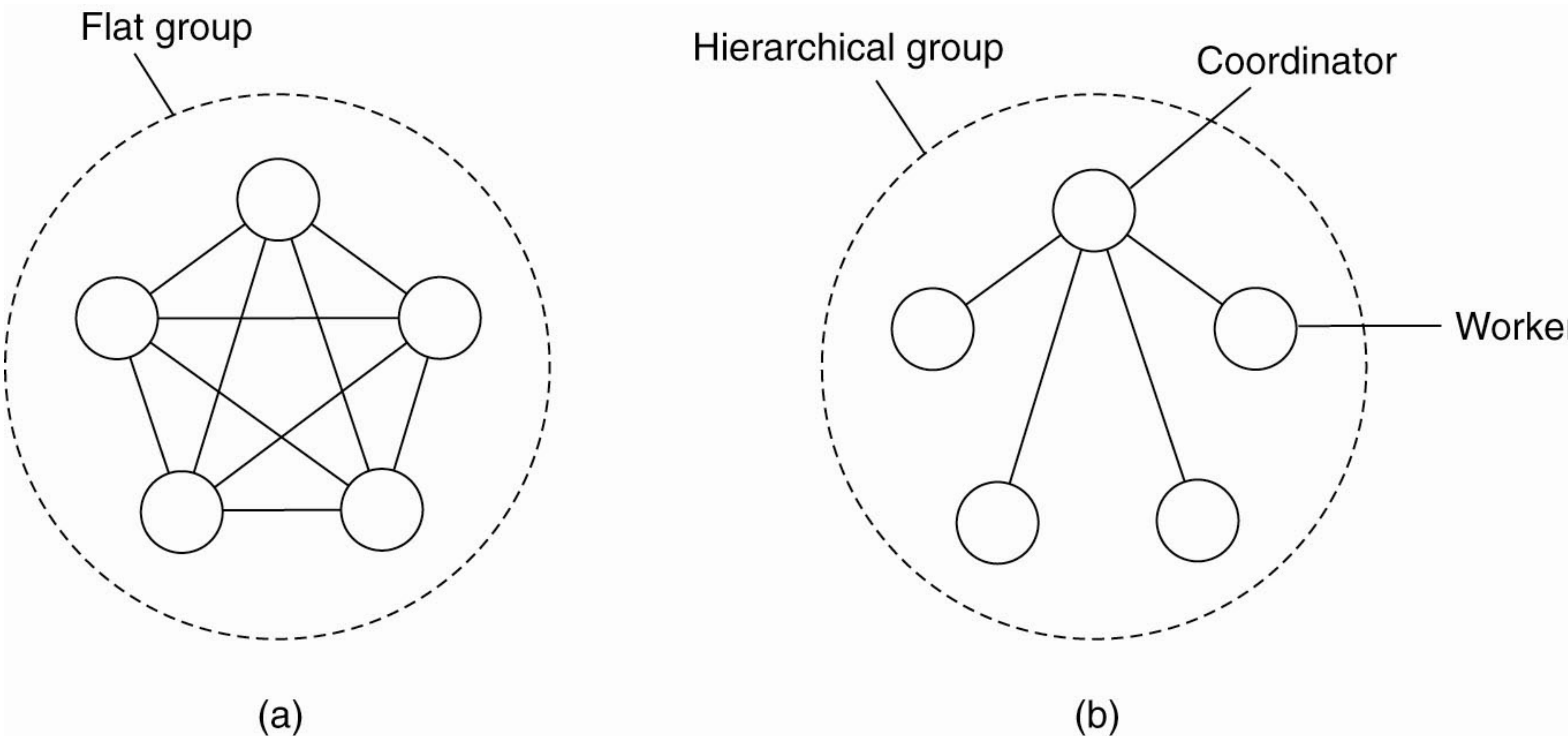
Failure Models

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure	A server fails to respond to incoming requests
Receive omission	A server fails to receive incoming messages
Send omission	A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure	A server's response is incorrect
Value failure	The value of the response is wrong
State transition failure	The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Failure Masking by Redundancy



Flat Groups versus Hierarchical Groups



(a) Communication in a flat group.
(b) Communication in a simple hierarchical group.

Dependability Properties: Dependability vs. Security

Dependability vs. Availability and Reliability Guarantees

- **Availability**
- **Reliability**
- **Safety**
- **Maintainability**

Availability and Fault-Tolerance and Conditions

Dependability vs. Security Guarantees

?

Dependability vs. Security

- Omission / Response Failures
  - A component fails to take an action that it should have taken
- Commission Failures
  - A component takes an action that it should not have taken, as a deviation to the expected correct behaviour

Not only accidentally …. But as Deliberate Failires

Dependability vs. Security

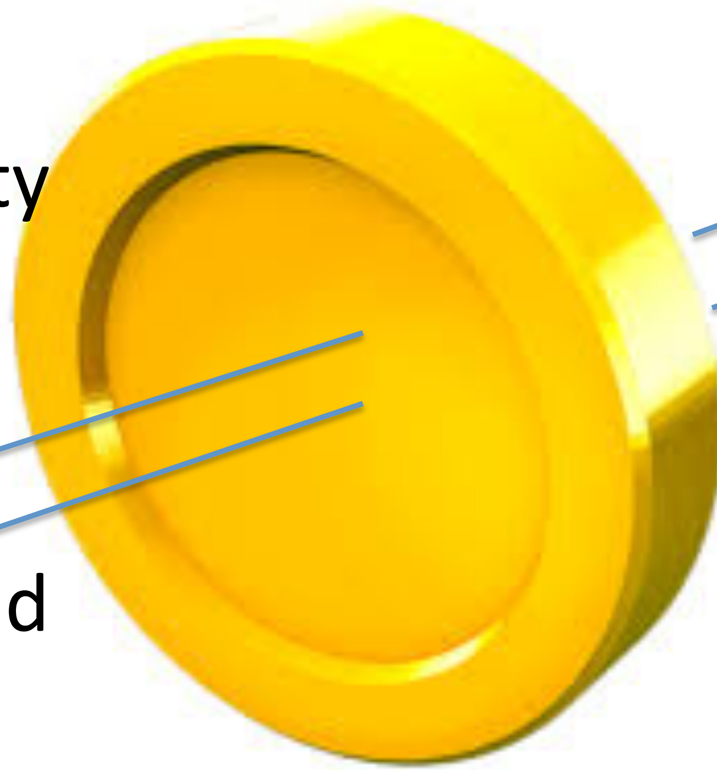
- Deliberate Failures, be they omission or commission failures, stretch out to the field of security
  - No accidental failures but induced failures by adveraries
- So …
  - There may actually be a “thin line” between Availability, Reliability and Security as dimensions of Dependability

Dependability vs. Security

- "Faces" in the same coin
- Challenge/Trend: Faces in the same solution ?

Availability and Reliability Properties

Fault Tolerance and Availability



Security Properties

Intrusion Tolerance

Fault Tolerance: Halting Failures

- Scenario:
- C no longer perceives any activity from C*
- A Halting Failure?
  - Distinguishing between **a crash or omission/ timing failure** may be impossible
  - In what circumstances ?

Fault Tolerance: Halting Failures

- In what circumstances ?
  - **Asynchronous system**: no assumptions about process execution speeds or message delivery times
    - → cannot reliably detect crash failures.
  - **Synchronous system**: process execution speeds and message delivery times are bounded
    - → we can reliably detect omission and timing failures.

Fault Tolerance: Halting Failures

- In practice we have **partially synchronous systems**:
  - most of the time, we can assume the system to be synchronous,
  - yet there is no bound on the time that a system is asynchronous

- → can normally reliably detect crash failures.

Fault Tolerance: Halting Failures

Assumptions we can make:

- **Fail-stop:** Crash failures, but reliably detectable
- **Fail-noisy:** Crash failures, eventually reliably detectable
- **Fail-silent:** Omission or crash failures: clients cannot tell what went wrong.
- **Fail-safe:** Arbitrary, yet benign failures (can't do any harm).
- **Fail-arbitrary:** Arbitrary, with malicious failures

Groups and Failure Masking

- **k-Fault-tolerant group:**
  - When a **group can mask any *k* concurrent member failures**
  - ***k*** is called **degree of fault tolerance**.

Dependable Systems

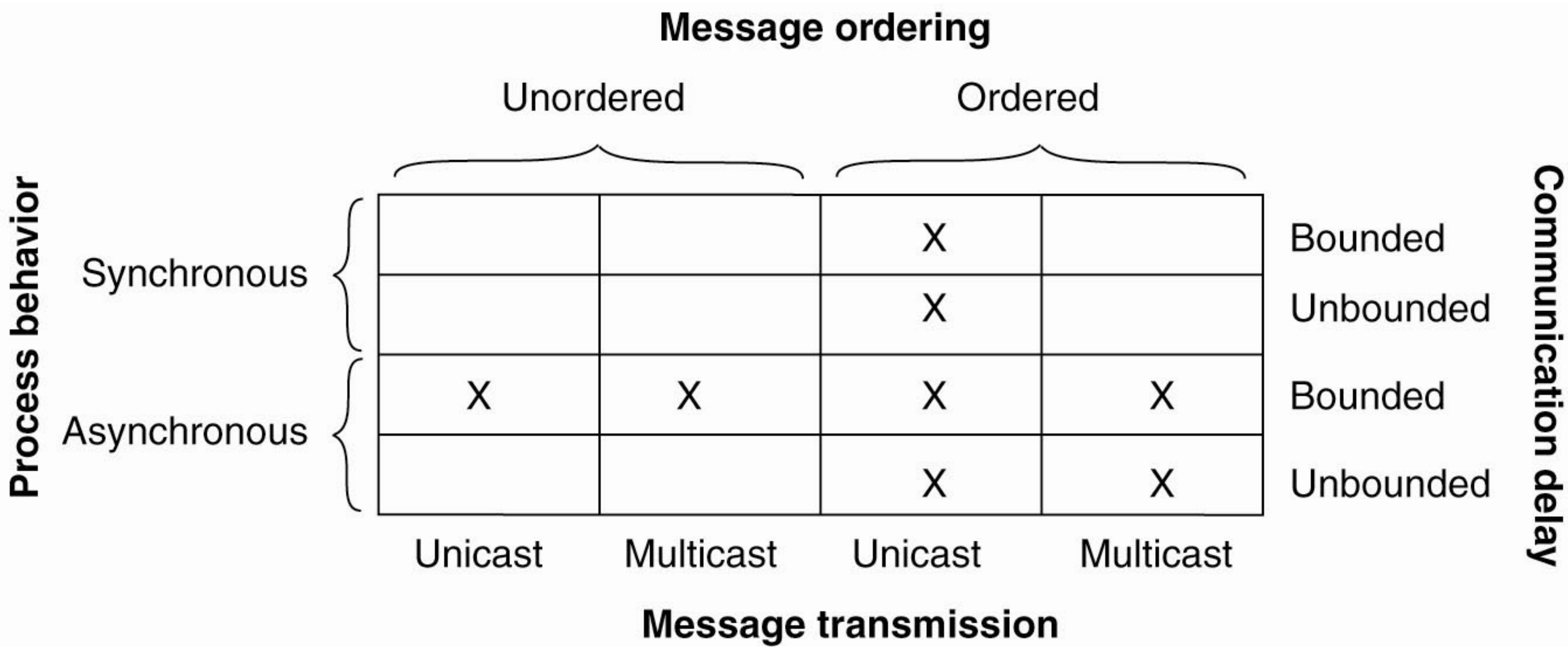
- Faut Tolerance, Agreement and Consensus
- // See also specific materials on the topic
  Week 2

Agreement in Faulty Systems

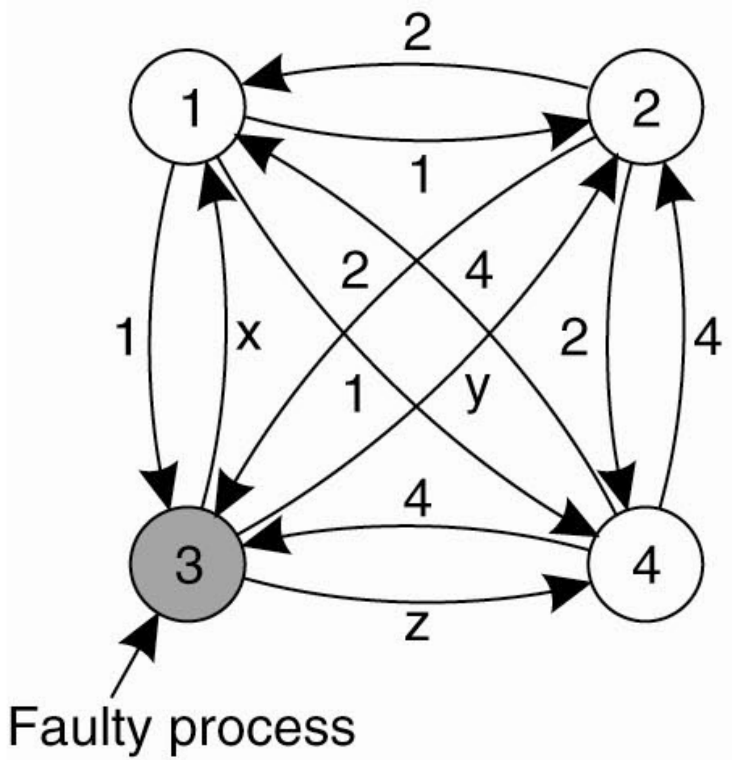
- Possible cases:

1. Synchronous versus asynchronous systems.
2. Communication delay is bounded or not.
3. Message delivery is ordered or not.
4. Message transmission is done through unicasting or multicasting.

Agreement in Faulty Systems (2)



Agreement in Faulty Systems (3)



(a)

- The Byzantine agreement problem for three non-faulty and one faulty process. (a) Each process sends their value to the others.

Agreement in Faulty Systems (4)

1	Got(1, 2, x, 4)
2	Got(1, 2, y, 4)
3	Got(1, 2, 3, 4)
4	Got(1, 2, z, 4)

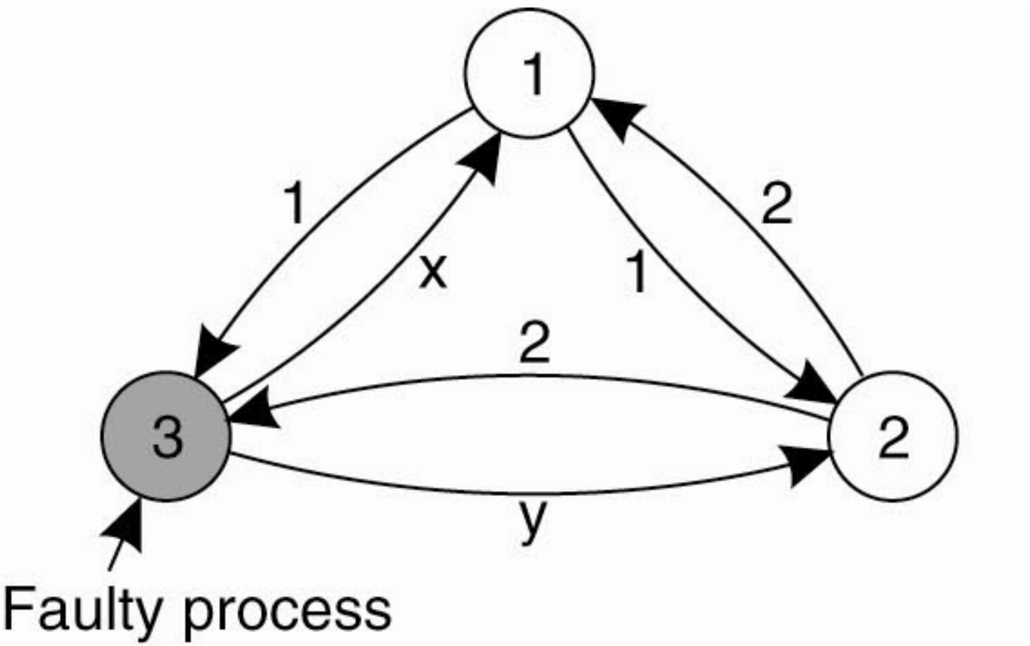
(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

- The Byzantine agreement problem for three nonfaulty and one faulty process.
  - (b) The vectors that each process assembles based on (a).
  - (c) The vectors that each process receives in step 3.

Agreement in Faulty Systems (5)



(a)

g. 8
nd

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

Groups and Failure Masking

How large must a k-fault-tolerant group be ?

- With halting failures (crash/omission/timing failures):
  - **we need *k+1* members**: no member will produce an incorrect result, so the result of one member is good enough.
- With arbitrary failures:
  - we need *2k+1* members: the correct result can be obtained only through a majority vote.

Groups and Failure Masking

Important:

- All members are identical
- All members process commands in the same order

Result:

- Only then do we know that all processes are programmed to do exactly the same thing.

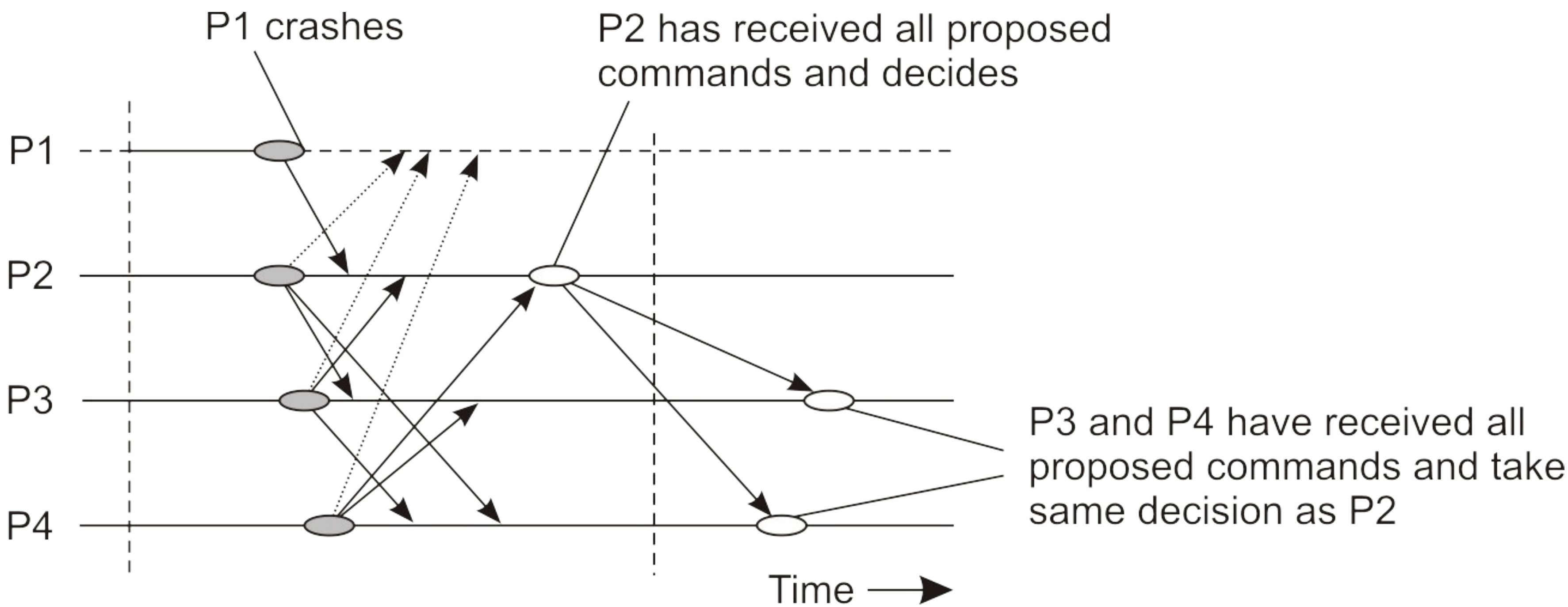
Observation

- The processes need to have consensus on which command to execute next

Flooding-based consensus

- Assume:
  - Fail-crash semantics
  - Reliable failure detection
  - Unreliable communication
- Basic idea:
  - Processes multicast their proposed operations
  - All apply the same selection procedure → all process will execute the same if no failures occur
- Suppose a process crashes before completing its multicast

Flooding-based consensus



Relevance for Intrusion Tolerance Protocols and Services

- Replication (ex., SMR)
- Consistency guarantees
  – Consistency Models, PAXOS, PAXOS-Variants
- Consistency vs. Performance
  – Role of Eventual Consistency Models

PAXOS

- **Assumptions** (rather weak ones):
  - An **asynchronous system**
  - Communication may be **unreliable** (meaning that messages may be **lost, duplicated, or reordered**)
  - **Corrupted messages are detectable** (and can thus be discarded)
  - All **operations are deterministic**
  - **Process may exhibit halting failures**,
    - but not arbitrary failures, nor do they collude.

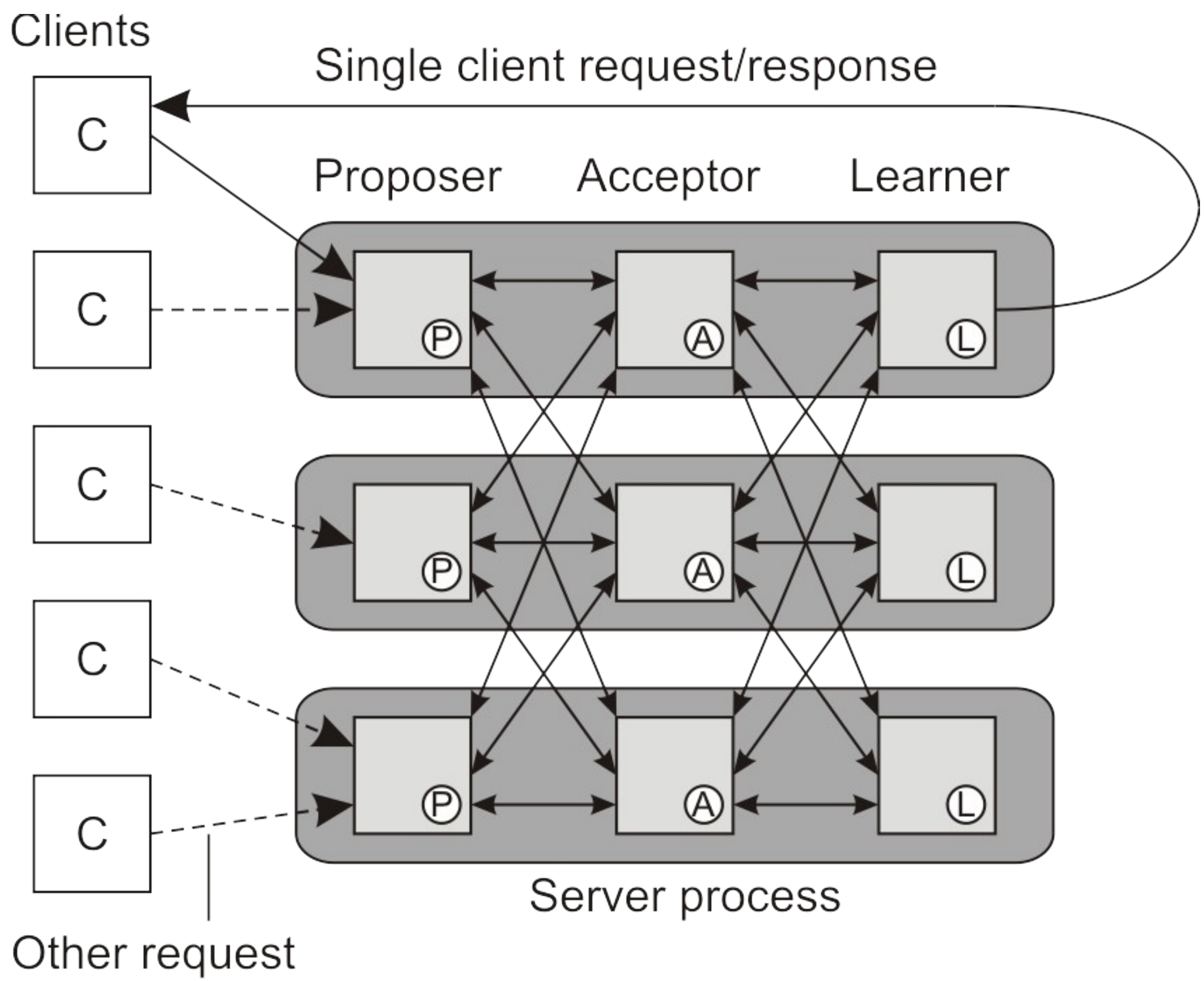
Essential PAXOS

- A **collection of (replicated) threads,** collectively fulfilling the **following roles**:
  - **Client:** a thread that requests to have an operation performed
  - **Learner:** a thread that eventually performs an operation
  - **Acceptor:** a thread that operates in a quorum to vote for the
  - **Proposer:** a thread that takes a client's request and attempts to have the requested operation accepted for execution

Essential PAXOS: Base Properties

- **Safety (nothing bad will happen):**
  - Only proposed operations will be learned
  - At most one operation will be learned (and subsequently executed before a next operation is learned)
- **Liveness (something good will eventually happen):**
  - If sufficient processes remain nonfaulty, then a proposed operation will
  - eventually be learned (and thus executed)

The PAXOS Environment …



Essential PAXOS

- New for some of you ?
- Review for others
  - => REVIEW next
  - More on WEEK 2